EEL 6764 001: Graduate Computer Architecture

Spring 2025

Instructor: Dr. Srinivas Katkoori

Homework 2

Hints/Solutions

1) (6 pts) Briefly (in 2-3 sentences each) describe six basic cache optimization techniques.

Solution: Six basic cache optimization techniques are described below:

- **Larger Block Size** – Increase the size of the blocks in accordance with the memory and communication bus sizes. This may reduce the required misses, but the cost of a miss will rise. The memory's latency and the buses' bandwidth both affect block size.
- **Larger total cache capacity to reduce miss rate** – To lower the cache's miss rate, expand the cache's size. Since there will be more records to search through in the cache, this optimization may result in a longer hit time.
- **Higher Associativity** – Increase the order of the associativity in the cache if the miss penalty is extremely high. Due to the necessity for more comparators, this may result in longer hit times and more power consumption.
- **Multi-Level Caching** – Cache at various levels should be used between the CPU and main memory. To maintain a low hit time, the cache closest to the CPU must be tiny, and each subsequent level of cache should strive to capture as many misses from the one before it as feasible.
- **Prioritize Read Misses overwrites** – Prioritize read errors above write errors, using various methods based on the write policy. If there are no read conflicts in a write-through cache with a write buffer, reading can be done before waiting for the write buffer to fill and flush. If a read miss results in a dirty block being replaced in a write-back cache, we can copy the dirty block to the write buffer so that the read can be done before the block is written to memory.
- **Avoid Address Translation** – By rearranging the structure between the CPU and main memory, you can try to prevent address translation during cache indexing in situations when virtual addresses are used. To do this, it could be possible to combine the L1 cache and TLB by placing them on the same level.

2) (8 pts) Briefly (in 2-3 sentences each) describe eight advanced cache optimizations techniques

Solution: Eight Advanced Cache Optimization Techniques:

- **Small and Simple L1 caches** – First-level caches should be a small size because of the pressure of a quick clock cycle and power restrictions. Moreover, using associativity at lower levels can shorten hit times and save energy. Since fewer cache lines must be accessed, lower degrees of associativity typically result in reduced power consumption.
- **Way Prediction** – To predict the way or block inside the set of the upcoming cache access to reduce hit time, extra bits are retained in the cache. The multiplexor is set early to choose the appropriate block as a result of this prediction, and just one tag comparison is carried out that clock cycle in parallel with reading the cache data. In the event of a miss, the following clock cycle will search the other blocks for matches.

- **Cache Pipelining** – The three-step process of addressing the tag memory using the index portion of the address, comparing the read tag value to the address, and configuring the multiplexor to select the right data item—if the cache is set up associative—is the important temporal path in a cache hit. This optimization merely involves pipelining cache access so that a first-level cache hit's effective latency can be several clock cycles, resulting in quick clock cycle times and high bandwidth but slow hits.
- **Multi-banked Caches** – We can divide the cache into discrete banks that can enable simultaneous access rather than treating it as a single monolithic block, as is done with DRAM. A good mapping of addresses to banks is to distribute the block's addresses among the banks in order to distribute accesses among all of them. It increases complexity, cost and power consumption.
- **Nonblocking Cache** – For pipelined computers that support out-of-order execution, the processor must halt on a data cache miss. By enabling the data cache to keep supplying cache hits even in the event of a miss, a nonblocking cache or lockup-free cache increases the potential benefits and increases cache bandwidth. By assisting during a miss rather than disregarding the processor's demands, this "hit under miss" optimization lowers the effective miss penalty.
- **Critical Word First/Early Restart** – This method is based on the observation that the processor typically only requires one word at a time from the block.
- **Critical word first:** To fill in the remaining words in the block, ask for the missing word first from memory and transmit it to the processor as soon as it is available. Then, let the processor carry on with its current task.
- **Early restart**: Retrieve the words in the normal order, but as soon as the block's desired word appears, transmit it to the processor and allow it to carry out the remaining instructions.
- **Merge Write Buffer** – Because every store needs to be transmitted to the next lower level of the hierarchy, write-through caches rely on write buffers to reduce miss penalty. The addresses of the new data can be compared to the addresses of valid write buffer entries as the buffer is filled. If so, fresh information is added to that record.
- **Hardware Pre-Fetching** – Using this method, things are prefetched before the processor demands them which reduces miss penalty or miss rate. Both data and instructions can be prefetched, either directly into caches or into a faster-accessing external buffer than main memory.

3) (10 pts) Effect of Locality – Solve problem B.1 on page B-60.
Solution:

$Average\ memory\ access\ time = (hit\ rate) * hit\ time + miss\ rate * miss\ time$ ....(1)

a) Substituting values in Equation (1), we get AMAT

= 97% * 1 + 3% * 110
= 0.97 + 0.03 * 110
= 0.97 + 3.3
= 4.27 cycles

b) Given, the size of array = 1GB, size of data cache= 64KB. Because of the randomness of access, the probability that an access will be a hit is equal to the size of the cache divided by the size of the array.

$$Hit\ rate = \frac{Size\ of\ the\ cache}{Size\ of\ the\ array} = \frac{64KB}{1GB} = \frac{2^6 * 2^{10}}{2^{30}} = 2^{-14}$$
$$\cong 6 * 10^{-5} \cong 0.00006103515$$

Therefore,

$$Average\ acess\ time = hitrate * 1 + (1 - hitrate) * miss\ cycles$$
$$= 6 \times 10^{-5} + (1 - 6 \times 10^{-5}) \times 110$$
$$= 109.99 \approx 110\ cycles$$

c) In part(b), when the cache is enabled, the average memory access is 110 cycles and when the cache is disabled the access time is 105 cycles. This shows that the enabling of cache memory increased the memory access time. If there is no locality, then there is no requirement of cache memory.

d) Let the missing rate be *x, then*

$$1 + (x)\ missrate \times 110\ miss\_cycles = 105$$
$$1 + 110x = 105$$
$$x = 0.945$$

So, after 0.945 miss rate, the cache use would be disadvantageous.

4) (10 pts) Fully Associative and 4-way Assoc. Cache – Solve problem B.2 on page B-60

Solution:

a)

Cache Block	Set	Way	Possible Memory Blocks
0	0	0	M0,M1,M2,.....,M31
1	1	0	M0,M1,M2,.....,M31
2	2	0	M0,M1,M2,.....,M31
3	3	0	....
4	4	0	....
5	5	0	....
6	6	0	....
7	7	0	M0,M1,M2,.....,M31

b)

Cache Block	Set	Way	Possible Memory Blocks
0	0	0	M0,M2,.......,M30
1	0	1	M0,M2,.......,M30
2	0	2	M0,M2,.......,M30
3	0	3	M0,M2,.......,M30
4	1	0	M1,M3,.......,M31
5	1	1	M1,M3,.......,M31
6	1	2	M1,M3,.......,M31
7	1	3	M1,M3,.......,M31

5) (10 pts) Cache Performance: Solve problem B.5 on page B-63.

Solution:

a) L1(instructions I-cache) miss time in L2
= Access Time of L2 + Transfer time
= 15ns + two L2 cycles = 15 + 2*3.759 = 22.5 ns
(two L2 cycles =32B in I-cache/16B width of L1-L2 bus)

L2 miss time in main memory
= Access Time of main memory + Transfer Time
=60ns + four memory cycles = 60 + 4*7.5 = 90 ns
(four memory cycles=64B in L2 cache/16B width of memory bus)

The average memory access time for instruction accesses
= (Hit rate * Hit time) + AMAT in L2 cache + AMAT in Memory + AMAT for L2 WB
= 0 + (miss rate of Icache)*(miss time required in L2)
+ (miss rate of Icache)*(miss rate of L2)*(miss time of L2)
+ (miss rate of Icache)*(miss rate of L2)*(dirty rate of L2)*(miss time of L2)
= 0 + 2%*22.5 + 2%*20%*90 + 2%*20%*50%*90
= 0 + 0.02*22.5 + 0.02*0.2*90 + 0.02*0.2*0.5*90
= 0 + 0.45 + 0.36 + 0.18
= 0.99 ns (1.09 CPU cycles)

b) Similar to the above formula with a difference: the data cache block width is 16B which takes one L2 bus cycles transfer (versus two for the instructions cache), so

L1(data D-cache read) miss time in L2
$= Access\ Time\ of\ L2 + Transfer\ time$
=15ns + one L2 cycle
=15 + 3.759
=18.759ns
(One L2 cycle=16B in D-cache/16B width of L1-L2 bus)

L2 miss time in memory = 90ns (which is derived in above equation)

The average memory access time for data reads
= (hit rate * hit time) + AMAT in L2 + AMAT in memory + AMAT for L2 WB
= 0 + (miss rate of Dcache read) * (miss time required of Dcache read in L2
+ (miss rate of Dcache read) * (miss rate of L2cache) * (miss time of L2)
+ (miss rate of Dcache read) * (miss rate of L2) * (dirty rate of L2) * (miss time of L2)
= 0 + 2%*18.759 + 2%*20%*90 + 2%*20%*50%*90
= 0 + 0.02*18.759 + 0.02*0.2*90 + 0.02*0.2*0.5*90
= 0 + 0.375 + 0.36 + 0.18
= 0.915 ns (1.01 CPU cycles)

c) Assume that writes misses are not allocated in L1, hence all writes use the write buffer. Also, assume the write buffer is as wide as the L1 data cache.

L1(data D-cache write) miss time in L2
= Access Time of L2+Transfer time
=15ns +one L2 cycle
= 15 + 3.759
= 18.759ns

L2 miss time in memory =90ns (which is derived in 5(a) question)

AMAT for data writes = (hit rate * hit time) + AMAT in L2 + AMAT in memory
+ AMAT for L2 WB

= 0 + (miss rate of Dcache write) * (miss time required of Dcache write in L2
+ (miss rate of Dcache write) * (miss rate of L2cache) * (miss time of L2)
+ (miss rate of Dcache write) * (miss rate of L2) * (dirty rate of L2) * (miss time of L2)
= 0 + 5%*18.759 + 5%*20%*90 + 5%*20%*50%*90
= 0 + 0.05*18.759 + 0.05*0.2*90 + 0.05*0.2*0.5*90
= 0 + 0.938 + 0.9 + 0.45
= 2.29ns (2.52 CPU cycles)

d) **Overall CPI (including memory accesses):**
= Base CPI + Instruction fetch CPI + Data read CPI + Data Write CPI
= 1.35 + 1.09 + 0.2*1.01 + 0.1*2.52 [load/store instructions are added to data read/write time]
=1.35 + 1.09 + 0.202 + 0.252
=2.894 cycles/instruction

6) (10 pts) Small vs. Large Cache Size: Solve problem B.12 on page B-65.
Solution:

a) Given, a direct mapped 8 KB cache has 0.22ns hit time and miss-rate-m1
4-way associative 64Kb cache has 0.52ns hit time and miss rate is m2, also given miss penalty is 100 ns

Let the access times for the above caches be T1 and T2

$$T1 = 0.22 + m1 \times 100$$
$$T2 = 0.52 + m2 \times 100$$

Larger caches tend to have longer hit times(T2) and smaller caches tend to have shorter hit times (T2) i.e. T1< T2

$$0.22 + m1 \times 100 < 0.52 + m2 \times 100$$
$$(m1 - m2) \times 100 < 0.32$$
$$(m1 - m2) < 0.32\%$$

It would be advantageous to use the smaller cache until the miss-rate of the smaller cache is up to 0.32% higher than that of the larger cache to reduce the overall memory access time.

b) $(T_s)_{10cycles}$ = 0.22ns + m1 * 10ns
$(T_L)_{10cycles}$ = 0.52ns + m1 * 10ns
To determine when the smaller cache is advantageous:
TS < TL.
0.22ns + m1 * 10ns < 0.52ns + m2 * 10ns
m1-m2 < 0.3/10
m1< m2 + 0.03

(Ts)1000cycles = 0.22ns + m1 * 1000ns
(TL)1000cycles = 0.52ns + m1 * 1000ns
To determine when the smaller cache is advantageous:
TS < TL
0.22ns + m1 * 1000ns < 0.5
2ns + m2 * 1000ns
m1-m2 < 0.3/1000
m1< m2 + 0.0003

The inequalities demonstrate that when the ratio of hit time advantage split by miss penalty is smaller, a smaller cache may be more favorable.

7) (10 pts) TLB hit/miss: Solve B.13 on page B-65.

Solution:

Page access trace

Virtual Page accessed	TLB (hit or miss)	Page Table (hit or fault)
1	Miss	Fault
5	Hit	Hit
9	Miss	Fault
14	Miss	Fault
10	Hit	Hit
6	Miss	Hit
15	Hit	Hit
12	Miss	Hit
7	Hit	Hit
2	Miss	Fault

8) (10 pts) Way Predicting Cache: Solve problem 2.18 on page 156.

Solution:

a) Let us consider the 4 – way associative cache.
Hit time = 3
Miss Rate = 0.33% = 0.0033
Miss Penalty = 20

Average memory access time = Hit Time + Miss rate * Miss penalty
= 3 + 0.0033 * 20
= 3.066 cycles

For the same cache, by using the way prediction, it is modelled as a direct mapped cache. This means that the hit time with correct prediction is the hit time of DM Cache. Its miss rate remains the same, that is 0.0033. The overall hit time has two components: hits with correct prediction and hits with incorrect prediction. Since the prediction accuracy is 80%, hit time is:

Hit Time = Accuracy * Hit Time DM + (1 – Accuracy) * Hit Time4-way

Accuracy = 80% = 0.8
Hit Time $_{DM}$ = 2 cycles
Hit Time$_{4\text{-way}}$ = 3 cycles
Hit Time = 0.8 * 2 + (1 – 0.8) * 3
= 1.6 + 0.2 * 3
= 1.6 + 0.6 = 2.2 cycles
Average memory access time = Hit Time + Miss rate * Miss penalty
= 2.2 + 0.0033 * 20
= 2.2 + 0.066
= 2.266 cycles = 1.13 ns.

b) The cycle time of the 64KB 4-way cache=0.83ns,
The cycle time of the 64 KB direct mapped cache=0.5ns,
=>0.83/0.5=1.66(or) 66% faster cache access.
c) With 1 cycle way misprediction penalty,
AMAT is 1.13 ns (based on part A question),
With a 15 cycle misprediction penalty, the
AMAT = 0.0033 * 20 + [0.8 * 2 + (1 – 0.8) * 15] * (1 – 0.0033)
= 0.066 + 4.6 * 0.9967
= 4.65 cycles
= 2.3 ns
Therefore, AMAT is increased by 2.3ns – 1.13 = 1.17ns.
d) Using the given data, Parallel access time = 1.59ns
Serialize access time 2.4ns
Ratio = 2.4/1.59 =1.509 ~ 1.51 =>
Serial access is 51% slower.

9) (10 pts) Critical Word First and Early Restart: Solve problem 2.20 on page 157
Solution:
a) The time to receive the first 16-byte block from the controller is 120 cycles.
Without the critical word first, the no. of cycles required would be 120+3×16=168 cycles. Assuming, that the first 16B block from the memory contains the requested data.
With the critical word first and early restart, the number of cycles required to service an L2 miss is 120 cycles.

b) To decide whether critical word first is more important to L1 or L2, its contribution to AMAT of L1 or L2 misses should be evaluated. Additionally, a reduction in miss service times (not the same as miss penalty) by critical word first and early restart should also be considered. If the reduction in miss service times provided by the critical word first and early restart is roughly the same for both L1 and L2 misses, then we need to consider the ratio of miss penalty over AMAT. These techniques help more for a cache with higher such ratio. Very often, such ratio for L1 is higher, they would likely be more important for L1 misses.

10) (10 pts) Multi-level Caches: Solve problem 2.22 on page 157.
Solution:

(a): 32KB L1; 8MB L2; Off chip memory

AMAT = (L1 misses) * 16 cycles + (L2 misses) * 200
= 100 * 16 + 10 * 200
= 1600 + 2000
= 3600 cycles

b. 32 KB L1; 512 KB L2; 8 MB L3; off-chip memory

AMAT = (L1 misses) * 4 + (L2 misses) * 16 + (L3 misses) * 200
= 100 * 4 + 50 * 16 + 10 * 200
= 400 + 800 + 2000
= 3200 cycles

c. 32 KB L1; 128 KB L2; 2 MB L3; 8 MB L4; off-chip memory

32 KB L1; 128 KB L2; 2 MB L3; 8 MB L4; off-chip memory.

AMAT = (L1 misses) * 2 + (L2 misses) * 8 + (L3 misses) * 16 + (L4 misses) * 200
= 100 * 2 + 80 * 8 + 40 * 16 + 10 * 200
= 200 + 640 + 640 + 2000
= 3480 cycles.

Case(b) with a 3-level cache is the best design when all access times are compared. Cache in 2-level can lead to numerous lengthy L2 accesses (1600 cycles).
Each level of the hierarchy may experience several pointless lookups in a 4-level cache.

11) (6 pts) Merging Write Buffer: Solve problem 2.21 on page 157.
Solution:

a) 16B, to match the level 2 data cache write path.
b) Assume merging write buffer entries are 16B wide. Because each store can write 8B, a merging write buffer entry would fill up in 2 cycles. The level-2 cache will take 4 cycles to write each entry. A non-merging write buffer would take 4 cycles to write the 8B result of each store. This means the merging write buffer would be two times faster.
c) With blocking caches, the presence of misses effectively freezes progress made by the machine, so whether there are misses or not doesn't change the required number of write buffer entries. With non-blocking caches, writes can be processed from the write buffer during misses, which may mean fewer entries are needed.

***